Process management

- Process or task: is an instance of a computer program that is being executed.
- Program is a passive collection of instructions, and a process is the actual execution of those instructions.
- Several processes may be associated with the same program, each would execute independently, either synchronously or asynchronously
- The operating system is responsible for the following activities:
  - Creation and termination of processes.
  - Allocate resources to processes
  - Share global files among processes
  - Protect processes
  - Process synchronization
- Dispatcher: is a program that move the processor from one process to another.

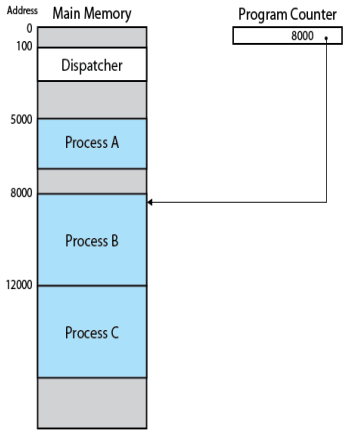


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

Process

- Consists of three components
  - An executable program
    - image in machine code
  - Associated data needed by the program
  - Execution context of the program
    - All information the operating system needs to manage the process

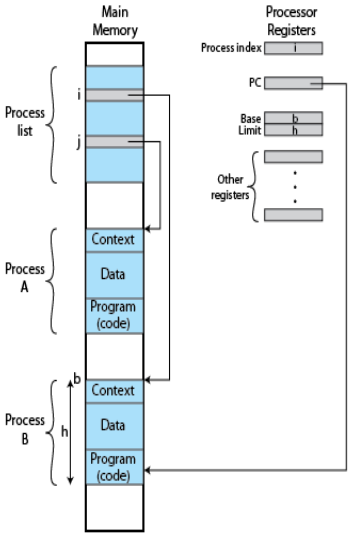
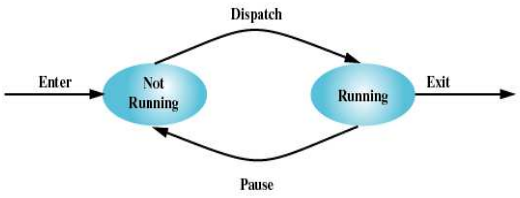


Figure 2.8 Typical Process Implementation

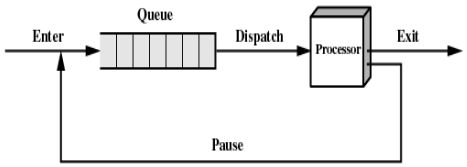
Process states

A two-state process model

- Process may be in one of states
  - Running
  - Not-running
- The processes that are not running kept in a queue waitir their turn to execute
- When a process is interrupted, it is transferred to the queue of waiting processes
- If the process has completed or aborted, the process is discarded
- The dispatcher then select a process from the queu to execute.



(a) State transition diagram



(b) Queuing diagram

Process Creation

Table 3.1 Reasons for Process Creation

New batch job	The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Process Termination

Table 3.2 Reasons for Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.

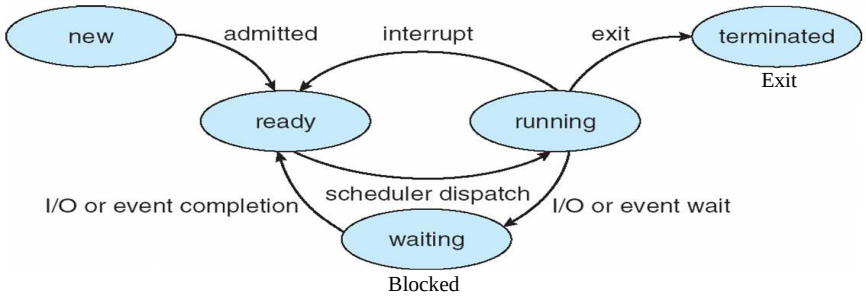
٣٣

Table 3.2 Reasons for Process Termination

Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

٣٤

A Five-State Model

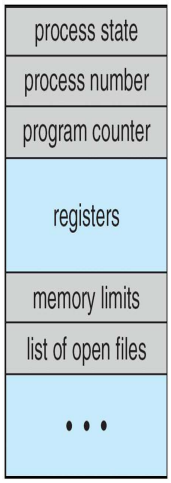


- **New**: The process is being created
- **Running**: Instructions are being executed
- **Waiting (blocked)**: The process is waiting for some event to occur
- **Ready**: The process is waiting to be assigned to a processor
- **Terminated (Exit)**: The process has finished execution

Process Control Block (PCB)

Information associated with each process

- Process state: may be new, ready, running, waiting
- Program counter: indicates the address of the next instruction to be executed for this process
- CPU registers: registers state information must be saved when an interrupt occurs, to allow the process to be continued later
- CPU scheduling information: includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- Memory-management information: include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system
- Accounting information: includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- I/O status information: includes the list of I/O devices allocated to the process, a list of open files, and so on

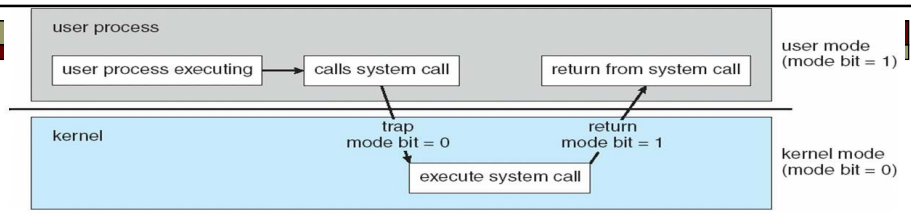


System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C -small and fast)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API (libraries- windows SDK) for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Many of the low-level functions in Windows were created using the C programming language. All of the DLLs in the Win32 API, and most of the kernel-level structures are implemented in C code.

Types of System Calls

- Process control - File management -Device management
- Information maintenance –Communications -Protection



Examples of Windows and Unix System Calls

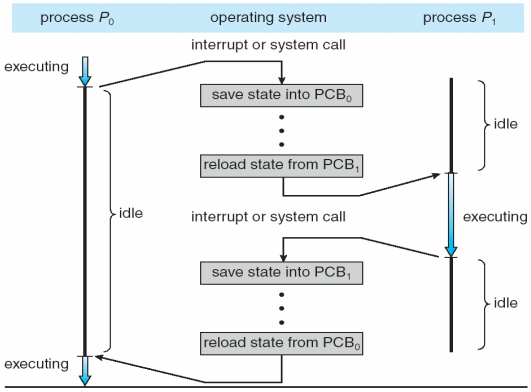
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

CPU Switch From Process to Process

•When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch

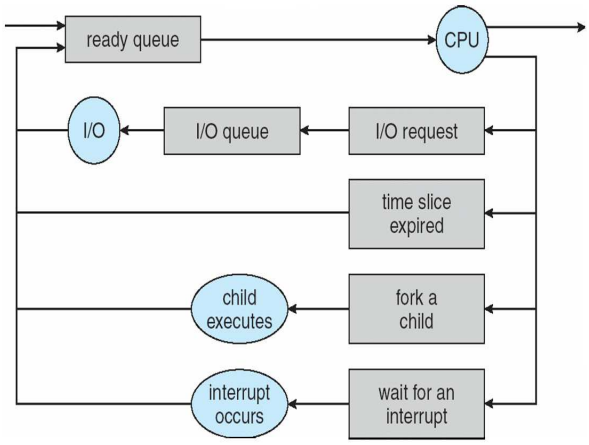
•Context of a process represented in the PCB. Context-switch time is overhead; the system does no useful work while switching

•Time dependent on hardware support



Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues



Process state transition

- Null → New
- New → Ready
- Ready→ Running
- Running → Exit
- Running → Ready
- Running →Blocked
- Blocked → Ready
- Ready→ Exit
- Blocked →Exit

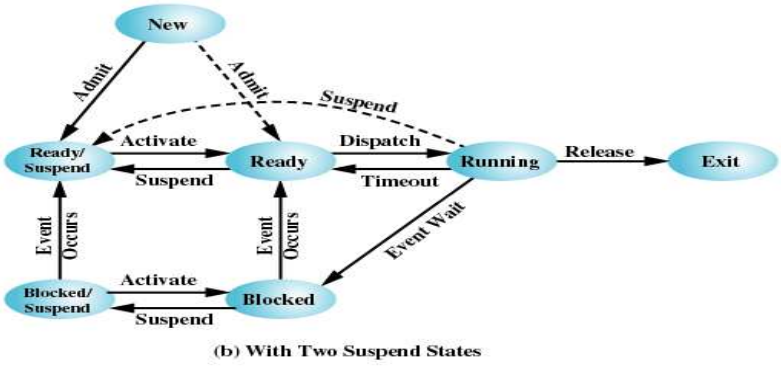
٤١

Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes suspend state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend
    - **Ready:** The process is in main memory and available for execution.
    - **Blocked:** The process is in main memory and awaiting an event.
    - **Blocked/Suspend:** The process is in secondary memory and waiting an event.
    - **Ready/Suspend:** The process is in secondary memory but is available for execution as soon as it is loaded into main memory

٤٢

Two Suspend States



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Reasons for Process Suspension

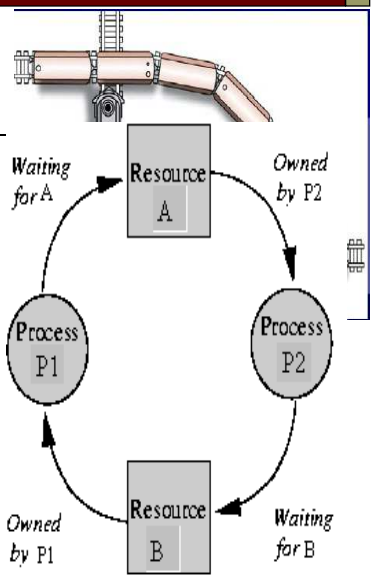
Table 3.3 Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.

Deadlock Problem

- Deadlock: A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set
- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

P2	P1
Request (A) Lock (A)	
	Request (B) Lock (B)
Request (B) Wait	
	Request (A) Wait



Resource allocation graph

Conditions for deadlock

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

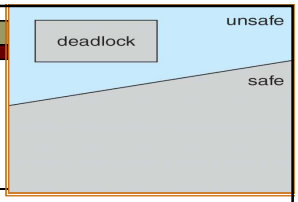
Deadlock Prevention

- Each process must request all its required resources R (CPU cycles, memory space, I/O devices) at one time and blocking the process until all requests can be granted simultaneously
- If a process holding certain resources is denied a further request, that process must release its original resources and if necessary request them again together with the additional resources
- Linear order of resource type

Resources categories

- Preemptable Resources: kind of resources that can be taken away from process without causing the computation to fail, for example is a memory allocation.
- Nonpreemptable Resources: kind of resources that can NOT be taken from its current owner process without causing the computation to fail, for example taken away CD ROM while a process burning a CD.

Safe State



- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in safe state if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes is the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$
- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on
- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.